PASS II AND RUNTIME ROUTINES OF IITPL COMPILER

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY
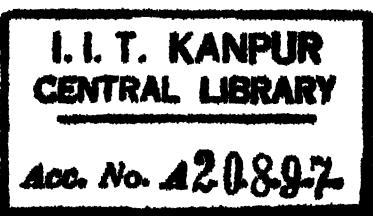
BY
MUKUL KUMAR SINHA

to the

EE-1971-M-SIN-PAS

CERTIFICATE

Certified that this work on "Pass II and
Runtime Routines of IIIPL Compiler" has been carried
out under my supervision and that this has not been
submitted elsewhere for a degree.

Dr. H.N. Mahabala
Associate Professor
Department of Electrical Engineering
Indian Institute of Technology
Kanpur

ACKNOWLEDGEMENT

I would like to express my gratitude to
Dr. H.N. Mahabala, the supervisor of the project,
for his inspiring and valuable guidance.

My special tnanks are due to Sri J.K. Sinha,
the co-worker of the project who showed his untiring
effort and extended his full help for the completion
of the project.

But the final achievement still remained a
mirage due to the system trouble (specially in DISK)
which stood in the way continuously for the last two
months.

Finally thanks are also due to Sri K.N.Tewari
for his excellent typing.

<div align="right">M.K. Sinha</div>

TABLE OF CONTENTS

SYNOPSIS

This project work has been done for the
implementation of IITPL, a language suitable for
system implementation, on IBM 7044. This compiler
is an out-core type, comprising of two passes.
Present work elaborates the working of Pass II
(of the compiler) and Runtime Routines (used
during execution).

The output of Pass II, the final coding
in MAP of a IITPL source programme is loaded to
the assembler, the output of which, inturn with
runtime routines, is fed as input to the loader
for the execution of the programme.

The programme is written in FORTRAN IV
with runtime routines in MAP, the assembly language
of IBM 7044.

CHAPTER I

INTRODUCTION

The purpose of the project "Implementation of IITPL on IBM 7044" is to select and implement a language suitable for system implementation.

This project work has been carried out jointly by the author and Mr. J.K. Sinha. Present thesis deals with the second part (last part) of the project. To understand the complete picture of the project, one is suggested to go through the thesis presented by Mr. J.K. Sinha.

In this part, the descriptions of the Intermediate Processor (which manages the provisions for subprogramme linkage and checks validity of inter-subprogramme control transfers), the coding done in PASS II, management of dynamic allocation, and runtime routines are given.

In PASS I of the compiler (See Appendix I) coding of all statements, except CALL statements and those GOTO statements, the coding of which is not possible in PASS I, is done. To facilitate the coding of CALL and above mentioned GOTO statements in PASS II; symbol table, ICALL table, and GOTO table are managed by PASS I (See Appendix II).

Symbol Table: It is a table of all the labels, variables declared in the subprogramme and block names. Particular labels or variables can be searched in symbol table by SYMTAB routine (See Appendix II).

ICALL table . ICALL table keeps all the cross refcrences between blocks.

GOTO Table: It is a table of strings of unfound labels of procedures. The first label of each string is the name of the procedure where rest lables of the strings are needed by GOTO statements.

For keeping the track of the blocks in PASS II, strings are provided by PASS I at coding points of PROCEDURE, BEGIN and END statements.

The PASS I output of the IITPL source program is on TAPE 0, which is taken as input for the PASS II. The final compiled output of the IITPL programme is written on TAPE 4 which is loaded for the execution.

CHAPTER II

INTERMEDIATE PROCESSOR

The functions of Intermediate Processor are the followings:

1) This checks the validity of subprogramme linkage and proper arrangements are done for inter-subprogramme transfer.

2) Possibility of recursion and invalid inter-subprogramme transfer is sensed.

Checking for Recursion:

With the help of ICALL table, the lowest level(L) procedure, called by any procedure is looked out first. If there are many procedures of lowest level (L), then the procedure which is topmost in the ICALL table is taken. Then a string of L level procedures elements with this procedure as HEADER, is formed. Each preceding link (or element) calls the following link. If any particular link (i.e. procedure) is found again, it shows the recursion. Thus is given error. The above mentioned string is formed in ICHECK table.

Similar operation is done, with next procedure of level L as header, until the checking with all the procedures of level L as header is done. Then similar checking is done with the precedures of level (L-1), (L-2),...2 respectively.

In this way all possible recursions are checked.

Provision for Inter-subprogramme Linkage:

In any subprogramme, one or more active blocks can be simultaneously ended by one single GOTO statement. It depends upon the path of activation of blocks and the availability in the active blocks of the label where the control wants to transfer.

Example·

```
        P1..PROCEDURE ,.
            CALL P2 ,.
        P2..PROCEDURE ,.
        L2..Stmt ,.
            BEGIN ,.
            CALL P3 ,.
        L3..Stmt ,.
            END P2 ,.
            CALL P3 ,.
        P3..PROCEDURE ,.
            GOTO L3 ,.
            Stmts
            GOTO L2 ,.
            Stmts
            GOTO L1 ,.
            END P3 ,.
        L1..Stmt ,.
        L2..Stmt ,.
        L3..Stmt ,.
            END P1 ,.
```

Procedure P3 can be activated in two ways:

1) P3 activated by P1 - In this case, all the three statements 'GOTO L3,' 'GOTC L2,' and 'GOTO L1,' will cause termination of the bl ck P3 and the control will go to the labels L3, L2 and L1 of procedure P1 for the corresponding GOTO statement.

2) P2 activated by P1, which activates BEGIN block, which in turn activates P3:-

Now the three GOLO statements of procedure P3 will adopt three different courses of actions. Each case is discussed below:

i) GOTO L3 - Since L3 is not available in P3, so P3 will be terminated and the control is now in the BEGIN block, which has a statement with label L3 and thus control will transfer to this L3.

ii) GOTO L2 - Since L2 is not available in P3, P3 will be terminated and the control is in BEGIN, where also L2 is not available and so this block will also be terminated and by tracing back the activation path, the control reaches the procedure P2, where L2 is found, thus the control transfers to that statement.

iii) GOTO L1 - Similar to L2, it will reach upto the procedure P2, but since L1 is not available even there, it will also be terminated and the control will transfer to L1 of the procedure P1, as if there were a 'GOTO L1' statement in the procedure P2.

It is quite clear from the previous example that points for the control transfer can only be provided after knowing all the possible paths of activation of blocks. That is why coding of C LL and GOTO (partially) is done in PASS II.

The ICALL table made in PASS I keeps the information of all the possible activations of blocks, which are used by Intermediate processor for providing points for the control transfer.

The Intermediate processor picks out the called procedure (say P) of lowest level (say L) from ICALL Table. If the procedure P is called by several blocks, then the topmost among them is taken. Let it be ICALL(L). Now calling block may be either a Procedure block or a BEGIN block.

Procedure Block: The availability of unfound labels(if any) of "Called Procedure" is checked in the procedure P1 calling it. If any particular label (L1) (or labels) is not found even in the calling procedure (P1) then it is treated as if 'GOTO L1' statement be in P1 and hence the label (L1) is attached to the string of unfound labels of calling procedure (P1) and the corresponding changes are done in GOTO table.

BEGIN Block: Firstly, with the help of symbol table, the path of activation of this BEGIN block is traced, until a PROCEDURE is found. These blocks are kept in

the block stack with assumption that the procedure is a
block of level 1.

If the BEGIN block is just under procedure there
will be only two blocks in the stack. But if the BEGIN
is under some other Begin and that under some other BEGIN..
and that under... and last one under procedure then all the
BEGIN blocks and the procedure will be in the block stack.

After arranging such a hypothetical block stack,
the availability of each unfound label of "Called Procedure"
is chedked.

1) If the label is not found in any of the block (even
in the procedure) of the block stack, then it is treated
just like the procedure case and the label is attached to
the string of unfound labels of the procedure in GOTO table.

2) If label is found in procedure or any BEGIN block of
block stack, the following information is kept in COLLECT
table, which is utilized at the time of coding in PASS II:

		L	VCB	
	VB	D	VL	

S 3   17 21   35

VCB — Value of calling BEGIN block

L   — Level of calling BEGIN block

VL — Value of Label

VB — Value of BEGIN block where label is found

D   — Level difference of calling BEGIN block and the
      BEGIN block where label is found.

Here 'Value' indicates the symuol table address.

After taking ablve action for each label of the called procedure, ICALL(L) is made zero.

The above action is taken again with the new ICALL table and this is done until all locations of ICALL table are made zero.

Now the expanded and new GOTO table is the final GOTO table which will be used in PASS II for coding.

PASS II

In the PASS II, the PASS I output, which is on the tape 0, is scanned statement by statement. If the first word of the statement is either of the five special characters (discussed below), it shows the presence of a string, for which the corresponding coding is to be done in PASS II.

Those special characters are

Characters	Octal Value	String of
∠	57	PROCEDURE
╲	76	BEGIN
Y	75	CALL
+++	77	GOTO/GO TO
;	56	END

Coding done by the compiler, for different types of strings is discussed below one by one.

1) PROCEDURE String

Example:

PASS I OUTPUT

```
        $IBMAP    FGH

        FGH       TRA       .I0001

        .V0001    SXA       ..IDX2,2

                  TSX       .SUBR2,2
```

```
                    PZE      3

                    PZE      5

                    PZE      3

          .I0001    BSS
```

```
| 00000 |    | S   T   A |
```

```
                    TSI      .RTRTN
```

where T — Type (0 for simple procedure,

4 for functional procedure).

S — Block number of procedure block.

A — Symbol table address of procedure name.

In this case, the first eight MAP instructions are
transferred in toto to the tape 4. The first word
ninth statement, indicates the presence of PROCEDURE s

The word, immediately after special character, keep
the symbol table address of the PROCEDURE name (i.e. TGH in
this particular case) in its address portion.

On encountering a PROCEDURE string in PASS II,
similar to PASS I, the procedure is kept in the Block
table.

With the help of the Symbol table address (A), the
number of variables declared in the procedure is taken and
it replaces the address portion of the second word of the
string. Now treating this word as an integer, coding

          CLA   =∅000001 000017

is done and the string is removed.

And the PASS II output will look as follows:

PASS II Output.

```
        $IBMAP    FGH

          FGH     TRA      .I0001

          .V0001  SXA      ..IDX2,2

                  TSX      .SUBR2,2

                  PZE      3

                  PZE      5

                  PZE      3

          .I0001  BSS

                  CLA      =Ø00000010 00017

                  TSL      .RTRTN
```

2) BEGIN String

Example:

PASS I OUTPUT

```
                  ..        ..

          M       CLA      5,1

                  ADD      2,1

                  STO      5,1
```



```
          IBLI·0

                  TRA      .I0007

                  ..        ..
```

where A - Symbol table address of BEGIN.

The first word of the above example, indicates the presence
of BEGIN string.

On encountering the BEGIN string, similar to PASS I,
the BEGIN block is kept in the Block table and the string
is removed   The PASS II output will look like as follows:

PASS II Output

```
            ..          ..
    M       CLA      5,1
            ADD      2,1
            STO      5,1
            TRA      .I0007
            ..          .
```

3) CALL String:

Example of PASS I output string and the expanded GOTO
table are shown below.

PASS I OUTPUT
```
                ..          ..
        .L0002  CLA*     2,1
                STO*     1,1
```

```
| 0 0 0 0 0 |  | 0 0 0 0 2 |   | 4 X X X X X |   | 0 0 0 0 0 5 |
| 1   3 0 36 |  | 4  1  36 |   | 0 0 0 0 0 7 |   | 0  7 0  36 |

| - 0 1 0 0 0 |
```

```
                TRA      .P0010
                ..          ..
```

<u>GOTO table:</u>

..		..		
..		..		
	21		0	15
FY				
M				
LSQ		S.A		
			1	
	6		1	20
	29		0	21
ZY				
N				
LSQ		S.A		
M				
LSQ		S.A		
			1	
	4		2	28
..		..		
..		..		

for LSQ, S.A see Appendix II.

With the help of these two, coding of CALL string is done.
It is also assumed that the CALL statement has appeared
in PROCEDURE FY. GOTO table shows that the label 'M' and
the label 'N' are not available in procedure 'ZY' and 'M'
is also unavailable in 'FY'; which shows that 'N' is the
label of any statement in procedure 'FY'.

Now with this CALL statement, return points are to be provided since labels 'N' and 'M' are unavailable in the procedure 'ZY'.

The fourth word of the string shows the number of arguments and the succeeding four words contain the information of the arguments (refer to <u>CALL argument characteristic word</u> in Appendix II).

Since label 'M' is also not avaiable in 'FY' procedure, so transfer is done to a new label calculated as follows:

$.C0000+C(ICALL(20))_{3-17}+$Position of label 'M' in

string of the procedure 'FY' (i.e. 1 in this case).
The statement, assigned this new label, will terminate the procedure 'FY' and will transfer the control to the upper block. The PASS II Output will look as follows:

<u>PASS II Output</u>

```
.L0002    CLA*    2,1
          STO*    1,1
          TSL     ZY
          TXI     *+6,,3
          PZE     3,1
          PZE     =7
          PZE     7,1
          TRA     N
          TRA     .C0007
          TRA     .P0010
```

4) <u>GOTO String</u>.

<u>Example</u>

   <u>PASS I OUTPUT</u>

```
            ..                    ..
                 ThA      K

         J      BSS


         00000#  000001  M#####

         K      CLA      1,1

            ..                    .
```

It is assumed that this GOTO string is found in
the procedure 'ZY' and the GOTO table of previous example
is also true for it.

The label 'M' is searched in PASS II also:

(a) If 'M' is found in the same block, the following
    coding is done

```
            TRA      M
```

(b) If 'M' is not found, it is picked out from GOTO table
    and its new label is calculated as in the case of the
    CALL string (i.e. .C0006 for 'M' in procedure 'ZY').
    And the coding will be:

```
            TRA      .C0006
```

(c) If GOTO string has come in BEGIN block 'P3' and if
    COLECT table is as shown below:

..		..	
	L	VCB3	
VB2	X	VIM	← N
..		..	

where

VCB3 — Symbol table address of BEGIN block 'B3'.

VB2 — Symbol table address of block 'B2' where
label 'M' is found.

VLM — Symbol table address of the label 'M'.

L — Level of block 'B3'

X — Level difference of blocks 'B3' and 'B2'.

the label 'M' is found in the JOLECT table and the
following coding is done:

$$\text{TRA} \qquad .\text{RC00n}$$

where 'n' is the location of the label 'M' in the JOLECT
table.

5) END String:

Example:

PASS I OUTPUT

```
            ..              ..
            CLA     3,1
            STO     5,1
           |00000;|
            CLA     =2
            TSL     .RTRTN
        ..              ..
```

END string may come either in BEGIN block or in PROCEDURE
block.

1) END Corresponding to BEGIN: From the block stack, the symbol table address of BEGIN (VCB) is searched. In COLECT area the presence of VCB as block, which wants label, is checked.

a) If it is there, the coding for all the desired labels are done with the help of their VB, X and VLM (see figure in the previous page). To encompass the set of statements, coding for the labels needed, a transfer instruction is given in the very beginning as:

$$\text{TRA} \quad .\text{ROOOm}$$

and at the end following instruction is generated:

```
.ROOOm    BSS
```

where 'm' = N-1, N is the COLECT table address of the BEGIN string.

The PASS II output will look as follows:

PASS II Output

```
                  ..                        ..
              CLA       3,1
              STO       5,1
              TRA       .ROOOm
    .ROOON    CLS       =iblno
              TSL       .RTRTN
              TRA       N
    .ROOOm    BSS
              CLA       =2
              TSL       .RTRTN
            ..                        ..
```

where 'iblno' is internal block number of the BEGIN block.

b)   If the BEGIN block is not found in the COLECT table
then no coding is done. The last block is erased from the
block stack and the string is removed.simply.

11) END Corresponding to PROCEDURE: From the block stack,
the symbol table address of the procedure is searched.

a) If the string of unfound labels is attached to the
procedure, the coding for all the unfound labels is done
similar to the BEGIN case but here GOTO table is taken
help of instead of the COLECT table. To encompass the set of
instructions, similar statements are generated here also.
It is assumed that the statement "END," corresponding to
the procedure 'FY' has come, then the PASS II output will
be as follows

P_SS II Output

```
              ..                              .
              CLA      3,1
              STO      5,1
              TRA      .G0001
.C0007        CLA      =2
              TSL      .RTRTN
              LXA      FY,4
              SXA      *+1,4
              LXA      **,4
              TIX      *+1,4,1
              SXA      FY,4
      .       TPA      FY
.G0001        BSS
              CLA      =2
```

Here the encompassing instructions have labels different from that of BEGIN case.

(b) If there is no string of onward labels attached to this procedure, only block stack entry is erased and no coding is done.

If the END statement is corresponding to the last procedure of the subprogramme, the control transfers to the END of the compiler and the PASS II output is now loaded on IBM 7044.

DYNAMIC ALLOCATION RUNTIME ROUTINES

The most beautiful feature of IITPL is that the declared identifiers and global identifiers have dynamic allocations, i.e. to say, identifiers declared in any block occupy the corresponding memory locations, only when, the block is active. As soon as the block ends (either by usual END statement, or by conditional or unconditional transfer to the point which is outside the block, at the time of execution), the allocated memory locations needed for this block are freed which may be used by another block, going to be active next.

Following example illustrates how dynamic allocation functions:

```
A..PROCEDURE OPTIONS (MAIN) ,.
    DECLARE (VA1, VA2, VA3, VA4) FIXED ,.
    Stmts
    CALL B ,.
B..PROCEDURE ,.
    DECLARE (VB1, VB2, VB3) FIXED ,.
    Stmts
    END B ,.
C,.BEGIN ,.
    DECLARE (VC1, VC2, VC3, VC4, VC5) FIXED ,.
    Stmts
    END A ,.
```

In the previous example procedure A calls procedure B.
Three variables are needed in this block (assuming that
no global identifiers are needed in block B). As soon as
procedure B is invoked during execution by the CALL
statement in procedure A, three locations are allocated
for B block and these locations are freed when control
transfers back in A block. Furthermore, when control
enters into C block (Begin block C), five memory locations
will be allocated to C block, three of them will be those
which were occupied by B block when it was active.

Thus overlapping of memory is done.

In IITPL this type of facility (i.e. dynamic
allocation) has been developed with the help of runtime
manager of variables (i.e. variable stack) and of active
blocks (i.e. active block stack).

The programme has been written in MAP, the
assembly language of IBM 7044. Here few names and
pointers used in the programme of runtime manager are
described:

ACTBLS (Active Block Stack).

This is a variable name which heads the active
block stack i.e. ACTBLS is the starting point of active
block stack. Each entry in the active block stack
occupies only one memory and consists of the following
information

	18	20	
IBLNO			SENTRY
	21		35

where IBLNO is the internal block number given to this block by the compiler to identify any active block during execution.

SENTRY is the starting entry in the variable stack after which data area for this block starts in variable stack.

..VAR   points out the starting point of the variable stack formed for the purpose of data allocation. The length of the variable stack depends upon the user's programme which is calculated during compilation for the worst case.

STPTKB works as a pointer and points to the starting point of current active block in variable stack.

NVARKB (Number of Variables in Current Block):

As the name suggests it gives the number of variables (or more accurately number of memory locations) needed by the current active block for data allocation.

To understand the function of the runtime manager and above mentioned names, let us consider the following example:

```
A..PROCEDURE OPTIONS (MAIN) ,.
    DECLARE (VA1,VA2,VA3,VA4,VA5(10)) FIXED ,.
    Stmts
    CALL C ,.
    CALL B ,.
```

```
      B..PROCEDURE ,.

          D.CLARE (VB1,V32,VB3,VL4,VB5)BIT(36) ,.

          Stmts

          VA1=VA5(2)+VA5(3)+VA3+VA4 ,.

          CALL C ,.

  BEG1..BEGIN ,.

          DECLARE B1(5,6,7) FIXED ,.

          Stmts

          CALL C ,.

          END A ,.

      C..PROCEDURE ,.

          DECLARE (C1) FIXED ,.

          DECLARE (C2,C3,C4,C5) BIT (50) ,.

          Stmts

          END C ,.
```

Initial instructions of any block (i.e. prologue of any
block) are those which load the block in active block
stack.

Referring to the present example, the prologue of
the block A will load the block A in active block stack.
Index1 and index2 act as pointers which give information
about starting point of data area in variable stack for
current active block and previous active block.

As the control enters into A block, the two stacks
(active block stack and variable stack) will look like

as follows:
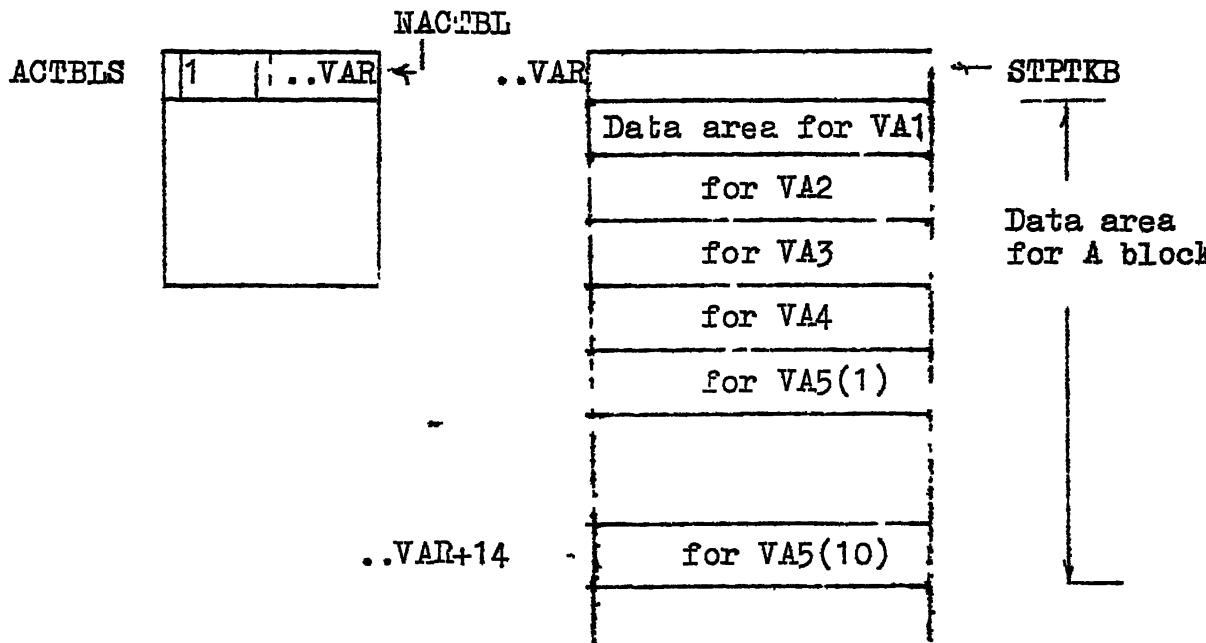
Here 1 in the decrement portion of ACTBLS shows the internal
block number of A while address portion of ACTBLS is filled
up by ..VAR, an entry in the variable stock just one
location before the starting point (..VAR+1 in this case)
of the data area for block A.

Pointers will have the values as follows:

NACTBL = 1   showing that number of active blocks is one.

NVARKB = 14 showing that number of memory locations
             needed by this block is 14.

index'1'= complement of ..VAR so that if any identifier
         is to be referred to,it will be referred to as:
         irpn,1
         where irpn is internal relative position number.
         irpn for VA1 = 1
         irpn for VA4 = 4
         irpn for VA5(7)=11 and so on.

index'2'= 0.

When procedure block C is activated by CALL statement in block A, block C is also loaded and entries of two stacks are:

ACTBLS	1	..VAR	..VAR	Data area for Block A
ACTBLS+1	4	..VAR +14	..VAR+14	
NACTBL			..VAR+15	Data area for Block C
			..VAR+19	
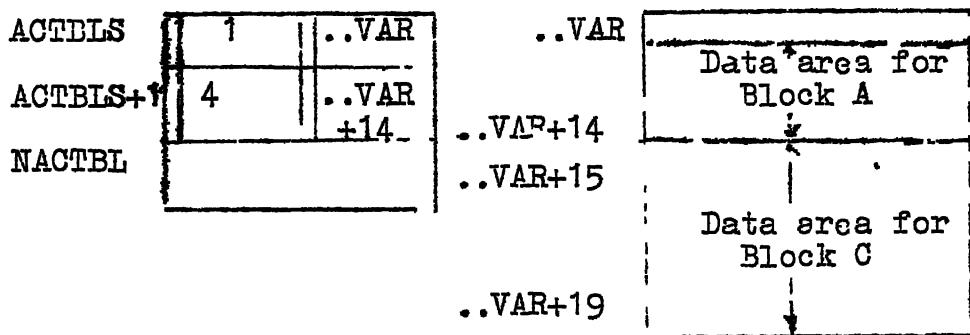
where 4 is the internal block number for block C.

NACTBL = 2   NVARKB = 5

index 1 = complement of (..VAR+14)

index 2 = complement of (..VAR)

When C block ends, this block is erased from the active block stack and the pointers are moved back accordingly.

Now   NACTBL = 1   NVARKB = 14

index 1 = complement of ..VAR

index 2 = 0

1	..VAR	..VAR	
			Data area for Block A
		..VAR+14	
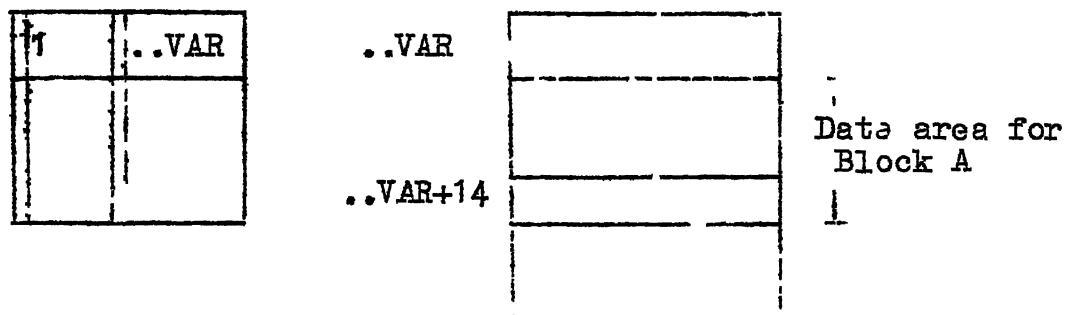
Similarly when B block is activated, different pointers will have the following values:

NACTBL = 2   NVARKB = 9

index '1' = complement of (..VAR+14)

index '2' = complement of (..VAR)

In this case NVARKB = 9, five locations for local identifiers and four for global identifiers (three for VA1,VA3 and VA4 and one for array name VA5).

It can be marked here that VA5 is an array name declared in A block and is needed by B block as global identifiers. Also in A block, ten locations are reserved (VA5(10)) for its allocation in variable stack while only one location is required in B block for the array name also if this array is global.

In this single location for array name, starting point of array name in A block is stored, so that actual address of any element of the array can be evaluated. Two stacks will look like:
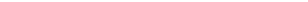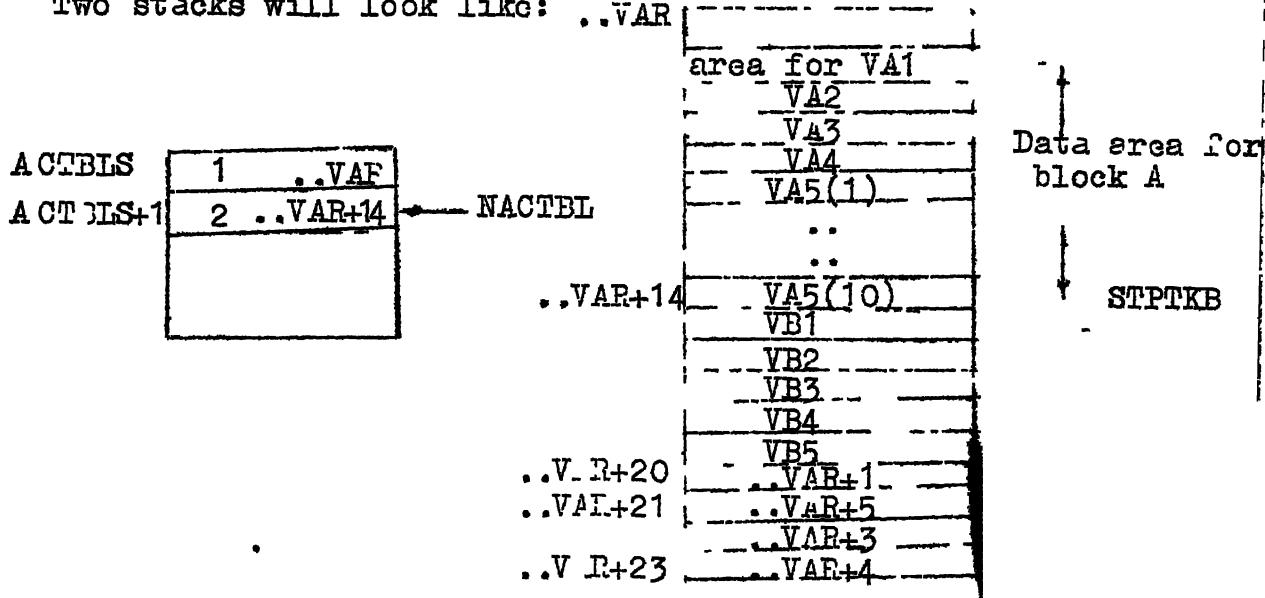
It can be noticed that ..VAR+20 contains the entry point
of the identifier VA1 i.e. (..VAR+1). Hence if VA1 is
referred in block B, it will be referred to as

OP-CODE' irpn,1

here'irpn'for VA1 in block B is 6.

Hence obviously, the indirect address of (6,1)
will refer to the address ..VAR+1 which, in turn, is
nothing but the entry for VA1. Similar is the case with
the other global identifiers.

Again when control enters into begin block the
stacks will be
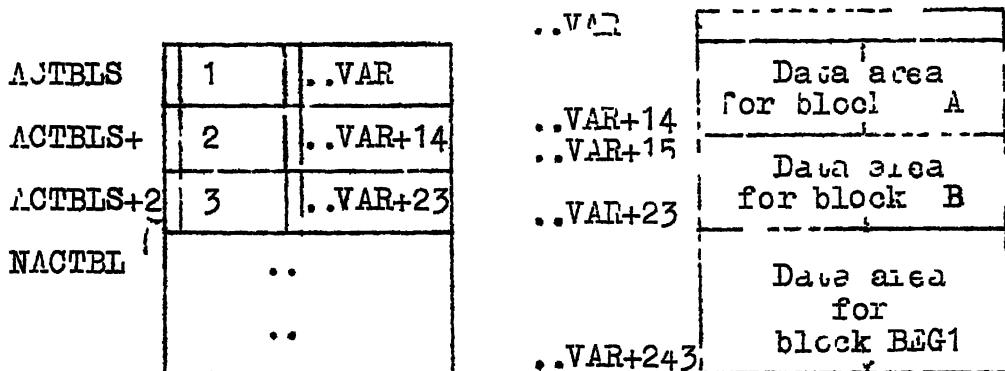
index '1' = complement of (..VAR+23)
index '2' = complement of (..VAR+14)

When CALL C in begin block BEG1 is executed,control is
transferred into C block and all the blocks are active
now. The different pointers will have the values·

NACTBL = 4   NVARNB = 210

STPTKB = ..VAR+243

index '1' = complement of (..VAR+243)

index '2' = complement of (..VAR+23)

and the two stacks will look like as follows:

ACTBLS	1	..VAR
ACTBLS+1	2	..VAR+14
ACTBLS+2	3	..VAR+23
ACTBLS+3	4	..VAR+243

NACTBL

..VAR	
	Data area for Block A
..VAR+14	
..VAR+15	Data area for Block B
..VAR+23	
..VAR+24	Data area for Block BEG1
..VAR+243	
..VAR+244	Data area for Block C
..VAR+248	

CHAPTER V
RUNTIME ROUTINES

.RTRTN   This is one of main runtime routines which does
the followings, depending upon the type of calling:

(1) It loads the block

(2) It erases the entry of current block from active
    block stack.

(3) It erases the entry of blocks unto the block mentioned
    by calling sequence.

(4) It loads the current block as well as checks the
    mode and type of arguments.

(5) It sends back the starting entry point of a block in
    Accumulator after fetching it from active block stack.

Calling Sequence of the Routine:

       Routine may be called in one of the following ways:

(1)   CLA = Number

           TSL   .RTRTN

Where Number = IBLNO (internal block number)*2**18

               + no.of locations needed in the block

i.e. contents of AC are:

IBLNO	0	no.of loc.

The decrement portion of AC denotes the internal block
number and the address portion contains the number of
variables (locations) needed for this block.

Function. In this case .RTRTN loads the current block in active block stack and sets the pointers NACTBL, NVARKB, STPTKB as well as index 1 and index 2. It also allocates memory locations needed for this block.

(2)   CLA = Number

    TSL   .RTRTN

where number is such that the content of AC is as follows:

| | IBLNO | 4 | No. of Loc. |

The only change in this type of calling with respect to the first type of calling is that TYPE = 4 i.e. $C(AC)_{18-20}$ is 4.

Function: In this case it is thought that this routine has been called from the beginning of either a parametric procedure block or from a functional block. In such a case it loads the current active block as well as compares for the proper mode and type between actual and formal parameters.

(3)   CLA =2

    TSL   .RTRTN

Function: In this case it is understood that the end of the current block has come (either due to the actual END statement or due to conditional or unconditional transfer). Hence the routine erases the entries for the current block from

block stack and the pointer in the variable stack is moved back to the block which is now active.

(4)   CLS   =Number

      TSL   .RTRTN

where the 'Number' is such that the C(AC) looks like:

1	Block No.	

S
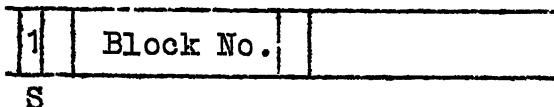
Only the decrement portion gives the information to the routine. Address portion of the Accumulator is of no importance hence it can be used for other purposes.

Function: The routine erases the entry from the active block stack upto the point where the block mentioned in decrement portion of AC is found in the active block stack.

(5)   CLA   =3

      LDQ   =Number

      TSL   .RTRTN

Here 'Number' denotes the block number, the starting entry point of which (in active block stack) is needed.

Function: The routine compares the block number sent in the MQ register with the blocks filled in active block stack. If it matches, the corresponding starting entry point is taken from the active block stack and put in the accumulator. This is useful in putting the entries in the variable stack for the global identifiers.

..FNCH This is a small segment of ..FTRTN routine and has
an entry point in this routine. Before entering into this
segment, it is understood that the control has been trans-
ferred from functional procedure. It checks for the valid
reference of the function and sends the mode of the function
procedure in runtime temporary storage.

..MDCH This routine is called by the following instructions:

$$TSX \quad ..MDCH,4$$

$$PZE \quad m,,tsp$$

where   'm' – denotes the mode of one operand

'tsp' – i.e. temporary storage pointer which

denotes the location where the mode of

second operand is filled.

Function: This routine compares the two modes and if they
match, control is returned to the calling point otherwise
error message is given. Such type of mode checking occurs
in case the expression is of the form .

⟨identifier⟩ ⟨operator⟩ ⟨function⟩

or function   operator   identifier

..MDCH routine can also be called by the following
instructions:

$$TSX \quad ..MDCH,4$$

$$MZE \quad tsp1,,tsp2$$

where   'tsp' – temporary storage point.

In this case modes are considered to be filled in the temporary storages mentioned in the decrement and address portion of the next location from the calling point (i.e. the location where the instruction is MZE tsp1,,tsp2). Such type of mode checking arises when the expression is of the form:

⟨function⟩ ⟨operator⟩   ⟨function⟩

RUNTIME ROUTINES

for subscripted variables (Array names)

Routines: .SUBR1, .SUBR2, .SUBR3

      Whenever an identifier, say NAME, is declared as
an array name, the declarative information (i.e. internal
relative position number 'irpn' of the first element of the
array and the upper limit of the subscript(s)) is stored as
a part of the object code which will be referred to as .Vn,
where 'n' is the internal number given to the array name.

      Storing of declarative information has the following
distinct uses by the object programme.

   (a) Wherever 'NAME' is used (in arithmetic statements,
I/O statements, etc.) it checks that the actual subscripts
lie within the bounds specified.

   (b) It calculates the absolute core address of the
array element specified by the given subscripts.

      Since .Vn is not a part of the executable object
code at the point where it is generated, a transfer ins-
truction is placed immediately before it to ensure that
it is not executed at object time.

      The object code generated for the array name 'NAME'
depends on the number of subscripts in the array and is as
follows:

```
        .Vn    SXA    ..IDX2,2
               TSX    .SUBPm,2
               PZE    irpn
               PZE    P
               PZE    Q  only for two dimensions
               PZE    R  only for three dimensions
```

.SUBRm   is one of the following routines depending upon
the number of subscripts.

   i)   .SUBR1   for single subscripted.

  ii)   .SUBR2   for double subscripted.

 iii)   .SUBR3   for triple subscripted.

'irpn' is the internal relative position number of the
first element of the array.

    During compilation, whenever 'NAME' appears with
subscripts, e.g. NAME(I,J,K), the following coding is
generated:

```
               TSX    Vn,4
               PZE    I
               PZE    J
               PZE    K
               PZE    DADRES
```

where 'DADRES' is the desired address where the absolute
address of the array element is to be put.

    If the subscripted name is referred to as the
global identifier, the following coding is generated:

```
              TSX     .V1,4

              PZE     I

              PZE     J

              PZE     K

              MZE     DADRES,,irpn
```

where 'irpn' denotes the internal relative position number
of the array name in this block.

   In all the cases index '2' contains the complement
of address of (.Vn+1) while index '4' contains the comple-
ment of the address of calling point.

   The calculation of the absolute address of an
element of the array name is based on the algorithm:

$$\text{address of } NAME(I,J,K) = \text{address of } NAME(1,1,1)$$

$$+ (I-1) + (J-1)*P + (K-1)*P*Q$$

Example:

```
   A..PROCEDURE OPTIONS (MAIN) ,.
      DECLARE (M,N1(5),N2(2,3),N3(4,5,6))FIXED ,.
      M = 2 ,.
      N1(2) = N2(M,3) + N3(M,5,M) ,.
      ...    ...    . .
```

Coding of the above statements will be as follows:

```
      $IBMAP A
              TRA     .I0001
      .V0001  SXA     ..IDX2,2
```

```
              TSX     .SUBR1,2
              PZE     2
              PZE     5
.V0002 SXA    ..IDX2,2
              TSX     .SUBP2,2
              PZE     7
              PZE     2
              PZE     3
.V0003 SXA    ..IDX2,2
              TSX     .SUBR3,2
              PZE     13
              PZE   ' 4
              PZE     5
              PZE     6
.I0001 BSS
              CLA     =2
              STO     1,1
              TSX     .V0001,4
              PZE     =2,,mode
              PZE     .T.+0
              TSX     .V0002,4
              PZE     1,1,mode          M
              PZE     =3,,mode
              PZE     .T.+1 contains address of N2(M,3)
              TSX     .V0003,4
              PZE     1,1,mode
              PZE     =5,,mode
```

```
PZE     1,1,mode

PZE     .T.+2   contains address of
                N3(M,5,M)

CLA*    .T.+1

ADD*    .T.+2

STO*    .T.+1

...     ...     ...
```

Instead of passing the subscripts by the instruction,
PZE Arg, if it is passed by the instruction, ONE Arg,
then the routines .SUBR1, .SUBR2 and .SUBR3 only check
if the subscripts are within bounds. In this case they
do not calculate the address of the element of the array.

.DORTN   This is a runtime routine used for handling
the modifications in control variables in a DO group
during execution. Calling sequence of .DORTN is as
follows.

```
TSX     .DORTN,4

CLA     c.v     control variable

STO     c.v

PZE     j       upper or lower bound of c.v

PZE     k       increment or decrement in c.v
                at each step.
```

.SUBPR   This is a runtime routine, which transfers the
the values of array elements from actual to formal para-
meter if both the parameters are array name and all
elements are to be transferred. This routine is used for
parametric procedure and function procedure.

.SBRST   This is a runtime routine, which transfers
values from formal parameter to actual parameter, if
both the parameters are array name and all elements of
which are to be transferred. This routine is called only
when a parametric procedure ends.

.STNDX   This is a runtime routine, which, when called,
stores the values of index registers. This is used only
for I/O statements.

..ESTT   This is a routine (runtime) which, when called,
restores the values of index registers. This is used only
for I/O statements.

Relational Operator Routine:

        There are six entry points in this routine·
(1) RO.EQ.   (2) RO.NE.   (3) RO.LT.   (4) RO.LE
(5) RO.GT.   (6) RO.GE.

These six entries are for relational operators .EQ., .NE.,
.LT., .LE., .GT. and .GE. respectively. .NL. and .NG.
are equivalent to .GE. and .LE. respectively.

        At each entry point, it checks if the condition is
true or not. In case it is true, $C(AC)_{p,1-35}$ is filled by
777777777777 with sign bit as plus; if the condition is
false, $C(AC) = 0$.

CONCLUSION

Like FORTRAN compiler, IITPL compiler is also an out-core compiler, except the Runtime routines, which are there, in the core, at the time of execution. The Runtime routines approximately take 900 memory locations. Besides this, the compiler takes the help of some FORTRAN routines (namely I/O routines, EXP1 ...) which also will occupy some memory space but the memory, occupied by these routines are programme dependent. Only those routines will be in the core which are required by the programme, otherwise they will remain outside.

The compiler output is in MAP (Assembly language of IBM 7044), hence the assembler and its IOCS occupy round about 6000 memory locations. So, out of total 32768 memory locations, approximately 25000 memory locations are free for the object programme.

Since the compiler is an out-core type, its symbol table is so designed that it causes the compiler to take the whole of memory. If some addition is to be done in the compiler, the size of the symbol table (which is at present 4000) can always be reduced with a slight change in the system.

All the three sections, Lexical, Pass I & Pass II, and Runtime routines, have been tested separately and successfully. Till the end, the programme, handling CALL & GOTO statements, in Pass II, could not be fully tested, hence all the three sections of the compiler were not combined.

Here an example is taken to illustrate the PASS I output of the source statement (1ITPL).

```
*PL/1
P1. PROCEDURE OPTIONS (MAIN) ,.
     DECLARE (A,B,C.D,E,I) FIXED ,.
     DECLARE F(5,6,7) FIXED ,.
 M...A=A+B ,.
 L..IF(A.EQ.B) THEN IF(C.GT.2) THEN IF(A.GE.C)THEN A=0 ,.
     ELSE ,.
     ELSE IF(A.LT.C) THEN C=5 ,.
     ELSE A=0 ,.
     CALL P2(A) ,.
     DO I=1, D WHILE A=3, 4 TO 7 BY B ,.
     A=A*(B+C/D**5)-8 ,.
     END/* THIS SHOWS THE END OF DO GROUP */ ,.
B1..BEGIN ,.
     DECLARE (L,M,N,P) FIXED ,.
     A=A+B ,.
     END B1 ,.
     GET EDIT (C,D,E) (F(6), X(3)) ,.
P2. PROCEDURE (G)/* SECOND BLOCK STARTS */ ,.
     DECLARE G FIXED ,.
     GOTO M,.
     RETURN ,.
     END P2 ,.
```

```
           G.. FORMAT3 F(6),X(1) ,.
               PUT EDIT(((F(A,B,C) DO A=D TO E
               BY I) DO B=3 TO E) DO C=I TO 6
               BY 3) (R(G)) ,.
               END P1 ,.
               *DATA
```

Coding   P1..PROCEDURE OPTIONS (MAIN) ,.
```
               DECLARE (A,B,C,D,E,I) FIXED ,.
   .           DECLARE F(5,6,7) FIXED ,.
```

```
                   $IBMAP    P1

                   P1        TRA      .I0001

                   .V0001    SXA      ..IDX2,2

                       -     TSX      .SUBR3,4

                             PZE      7

                             PZE      5

                             PZE      6

                             PZE      7

                   .I0001    BSS

                   00000A    XXXXXX   A string which looks like
```

	IBLNO	O	VALUE
3	17	21	35

```
                             TSL      .RTRTN
```

H..A=A+B,.

```
                       M     CLA      1,1

                             ADD      2,1

                             STO      1,1
```

L..IF(A.EQ.B) THEN IF(C.GT.2) THEN IF(A.NE.C) THEN A=0,.

```
        L       CLA     1,1
                SUB     2,1
                TSX     R0.EQ.,4
                TZE     .E0001
                CLA     3,1
                SUB     =2
                TSX     R0.GT.,4
                TZE     .E0002
                CLA     1,1
                SUB     3,1
                TSX     R0.NE.,4
                TZE     .E0003
                CLA     =0
                STO     1,1
                TRA     .F0001
```

ELSE,.
ELSE IF(A.LT.8) THEN C=5,.
ELSE A=C ,.
ELSE ,.

```
        .E0003  TRA     .F0001
        .E0002  BSS
                CLA     1,1
                SUB     =8
                TSX     R0.LT. ,4
```

```
                    TZE      .E0004

                    CLA      =5

                    STO      3,1

                    TRA      .F0001

        .E0004 BSS

                    CLA      3,1

                    STO      1,1

                    TRA      .F0001

        .E0001 TRA      .F0001

        .F0001 BSS
```

CALL P2(A) ,.

000001	000002	P2##00
000001	XXXXX	-01000

where 'XXXXX' is current
characters the ' ' .

DO I = 1,D WHILE A=3,4 TO 7 BY 3 ,.

A=A*(B+C/D**5)-8 ,.

END /* THIS SHOWS THE END OF DO GROUP */ ..

```
                    CLA      =1

                    STO      6,1

                    TSL      .D0001

                    CLA      4,1

                    STO      6,1

                    CLA      1,1

                    SUB      =3

                    TSX      R0.EQ.,4

                    TZE      *+2
```

```
            TSL       .D0001
            CLA       =4
            STO       6,1
            CLA       =7
            STO       D.0001+3
            CLA       2,1
            STO       D.0001+4
.W0001      TSL       .D0001
D.0001      TSX       .DOFTN,4
            CLA       6,1
            STO       6,1
            PZE       0
            PZE       0
            TZE       E.0001
            TRA       .W0001
.D0001      TRA       *-*
```

* Coding of next statement starts here i.e.
Assignment statement in this case.

```
            LDQ       4,1
            MPY       4,1
            STQ       .T.+0
            MPY       .T.+0
            MPY       4,1
            STQ       .T.+0
            LDQ       3,1
```

```
            PXD       ,0
            DVP       .T.+0
            LLS       35
            ADD       2,1
            LRS       35
            MPY       1,1
            LLS       35
            SUB       =8
            STO       1,1
      * Coding of END statement
            TRA       .D0001
      E.0001 BSS

B1..BEGIN ,.
    DECLARE (A,M,N,P) FIXED ,.
    A=A+B ,.
    END B1 ,.

            00000      XXXXX       A string which looks like
```

		BINO	O	VALUE
3		17	21	35

```
      B1    TRA       .I0002
            .I0002 BSS
            TSL       B.0002
            CLA       1,1
            ADD*      5,1
            STO       1,1
      00000;
```

```
            STO     =2
            TSL     .RIRTN
            TRA     .B0002
B.0002  TRA     **
            CLA     =000000200C005
            TSI     .R TN
            CLA     =3
            LDQ     =1
            TSL     .RTL TN
            ADD     =2
            STO     5,1
            TRA     B.0002

.B0002  TSB
GET EDIT(C,D,E) (F(5),X( ) ,.
            TSI     STTTX
            TSX     TSHIO.,4
            STO     TLLO5
            PZE     nS
            TSL     .S T.
            TSL     HILIO.
            STO     3.1
            TSL     HNLIO
            STO     4,1  .
            TSL     HNLIO.
            STO     5,1
            TSX     RTNIO.,4
            TRA     mE
    nS      TSX     IORTC.,4
            PZF     6
```

```
                    PZE     3

                    T       IOHEF.

            nE      BSS                     where 'nE' is internal
                                            name given by the
                                            compiler, and 'm'=n+1
```

P2..PROC DUMP(C) ,.

   DECL nS C 'CARD ,.

```
                    J E     .P0003

            P2      M .     '.

                    LAJ     *-1,4

                    TXA     .I0003

            .I000J  BSS

            00:000^ XXXXXX  string similar as mentioned
                                            earlier.

                    TSL     .TT.TN

                    TXI     *+2,,1

                    PZE     1

                    CLA     1,4

                    PLT     1,4

                    ANA     =Ø000000077777

                    ANA     =Ø000000277777

                    ORA     =Ø050000000000

                    SLW     *+1

                    CLA     **

                    STO     1,1

                    TSL     B.0003
```

GOTO M ,.

```
            00000M  00001   MØØØØ^
```

RETURN ,.

```
                    TRA      A.0003
END P2 ,.
                _____
                00000;

           A.0003 CLA      1,4
                    PLT      1,4
                     M       =∅000000077777
                    _NA      =∅0000603277177
                    ORA      =∕050000000000
                    SLW      *+2
                    CLA      1,1
                    SLO      **
                    CLA      =2
                    PSL      .RTRTN
                    TRA      P2
           B.0003 TRA      **
                    TRA      B.0003
           .P0003 BSS
G..FORMAT(3 F(6),X(1)) ,.
                    TRA      nE
           C         AXT      3,2
                    TSX      IOHIC.,4
                    PZE      6
                    TSX      IOHXC.,4
                    PZE      1
                    TRA      IOHEF.
           nE       BSS
```

PUT EDIT(((F(A,B,C) DO A=D TO E BY I) DC B=3 TO E) DO C=I TO 6

BY 3) (R(G)) ,.

```
                TSL     STNDX.
                TSX     STHIO.,4
                TWO     FIL06.
                PZE     2S
                TSL     RESET.
                LDQ     6,1
                ST4     3,1
        .S0002  BSS
                LDQ     =3
                STQ     2,1
                CLA     5,1
                PAX     0,4
                SXD     .S0003,4
        .S0004  BSS
                LDQ     4,1
                STQ     1,1
                CLA     5,1
                PAX     0,4
                SXD     .S0005,4
                CLA     6,1
                PAX     ,4
                SXD     .T0003+2,
                LDQ     3,1
                VLM     =983040,,1·
```

```
        AIS    15
        LDQ    2,1
        VLA    =165840,,15
        ?I?    1,1
        SUB    ..IDX1
        ???    ,2
        CLA    6,1
        PAX    ,4
        SXD    . V001,.
.S0006 ???
        ???    .V0001,1
        ? 4    1,1
        ? ?    ,1
        0??    5,1
        ? ?    -20,?
        .ST.   .??IL?.
.N0??? ?X?     +1  ,
.T0003 ?L      1,1
        PAX    0,4
        TXI    +1,4,
        PXA    0,4
        STO    1,1
.S0005 TXL     .S0006,4,**
.T0002 CLA     2,1
        PAX    0,4
        TXI    *+1,4,1
        PXA    0,4
```

```
                    STO     2,1
            .S0003  TXL     .S0004,4,**
            .T0001  CLA     3,1
                    PAX     0,4
                    TXI     *+1,4,3
                    PXA     0,4
                    TXL     .S0002,4,6
                    TSX     RTNIO.,4
                    TXL     ..IDX2,2
            2S      EQU     G
END 21 ,.                   00000;
                    TRA     S.JXIT
                    ENTRY   ..VAR
                    .XMIN   ..IDX2
                    EXTERN  .RORIN
                    EXTERN  .RTRIN
                    EXTERN  .SSUBR5
                    EXTERN  .DORTH
            ..VAR   BSS     223
            .T.     BSS     1
                    END     P1
            $ENTRY
```

Note:

.SUBRm - This is the runtime routine for 'm' subscripted
variable. It checks whether the subscripts exceeds its
upper limit or not. If any one of the 'm' subscripts
exceeds its upper limit, it gives error in execution time.

.ROUTN - This is the relational operator runtime routine
which sends zero in AC if the condition is false otherwise
it puts 777777777777 in AC.

.RTRTN - This routine does the following:

   i) if C(AC) is greater than 3, then it assumes that
contents of Accumulator is in the following way:

Block No.	Type	No. of variables needed

   If Type = 1 - then it loads the current block in the
                 active block stack (ACTBLS) and moves
                 the pointer (Index1) correspondingly.

        = 4 - It assumes that current block is a
                 parametric procedure block or functional
                 procedure block. It checks the nodes of
                 the arguments and loads the current block.

   ii) If the C(AC)=3 - it assumes that block no. has
been sent in MQ. It then takes out the starting pointer
in the variable stack (..VAR) for this block and puts it
in AC and returns to the calling point.

   iii) If the C(AC) = 2 - it erases the current
block entry from ACTBLS (Active Block Stack).

.DORTN - This routine checks whether the control variable
has reached its limit value or not. If not it sends'1' in
AC otherwise puts zero in AC and returns.

..IDX2 - This saves the content of index2.

ICALL Table:

In PASS I, when any procedure is called from any block; level and the symbol table address of the "Called Procedure" and that of 'Calling Block' is kept in a pair of words of ICALL table as shown below:

3		17 21		35
L	VCLP	1	VCLB	
LCP				

where

    L   - Level of "Called procedure".

    1   - Level of "Calling block".

    VCLP - Symbol table address of 'Called procedure'.

    VCLB - Symbol table address of "Calling block".

    LCP - Label name of 'Called procedure'.

It is not always possible to find the "Called procedure" in symbol table, in that case $C(first word)_{S-17}$ is filled with zero. When a PROCEDURE statement is encountered in PASS I, it is checked whether the name of the new procedure appears any where in the ICALL table undefined. If yes then L and VCLP is filled at that time.

Before filling information in the ICALL table, following things are kept in mind:

    i) Repetition is avoided.

    ii) (a) If "called procedure" is already in symbol table

the following condition is to be satisfied:

L-1 = 0 or 1

(b) If "called procedure' has not come atall before the appearance of CALL statement, the validity of above condition is checked when PROCEDURE statement with the label of 'Called procedure" appears.

One hundred locations are reserved for ICALL table and the variable "NOCALL" is used as pointer of ICALL table.

CALL Argument Characteristic Word

M	IRPN	T	BIT
3	17	21	35

where IRPN = Internal position number of the variable
or temporary (when the argument is expression).

BIT = Number of bits required in the case of bit
string and character string, otherwise 0.

T = 0 simple variable

= 1 Single subscripted

= 2 Double subscripted

= 3 Triple subscripted

= 4 Temporary (in case of expression)

= 5 Constant

= 6 Functional argument

when T=5 'IRPN' points the number of words following the characteristic word which store the constant.

M = 0 Label variable

= 1 Simple and **subscripted variable**

= 2 Bit string variable

= 3 Character string variable

= 5 External fixed variable

= 6 External bit variable

= 7 External character variable

GOTO Table

GOTO table is is the table of unfound labels of procedures. All unfound labels of any particular procedure are kept as an element of the string, whose HEADER keeps the name of the procedure and information about the next link of the string.

PROC 1		1
NPROC 1	FLINK1	2
LABEL		3
L.S.Q	S.L	
.. ..	.. ..	
	0	n
	0	n+1
PROC 2		n+2
NPROC2	NLINK2	n+3
.. ..	.. ..	
	1	n
PROC2		m+1
PROC1		m+2
NPROC3	NLINK3	m+3
.. .	.. ..	

NLAB

where   NPROC - Table address of the first element of one

string, which starts just after this string

section is over.

NLINK - Table address of the HEADER of the next link

of the very PROCEDURE string.

SA   - Symbol table address of the label.

L.S.Q - Level Sequence - number

Before storing the unfound label and its characteristics, it is always checked whether it is already an element of the string of unfound labels of the procedure.

If END statement corresponding to any procedure is encountered and if this procedure is having a string of unfound labels, then it is permanently closed it storing 1 and procedure's label in the last two consecutive locations of the string of the GOTO table and running procedure is rubbed off from IGBLK table.

If any procedure starts before permanent end of the last running procedure, then the GOTO table string of the previous procedure is temporary closed with 0 and 0 in next two free locations and then HEADER for new string is kept thereafter.

As shown in example above:

NPROC1 = Table address of PROC2 i.e. n+2.

NLINK1 = Address of next link of PROC1 i.e. n+2

NPROC2 = Table address of next procedure(i.e. of

PROC1) i.e. m+2.

NLINK2 = 0

IGBLK Table:

This is the procedure block table used for handling the GOTO table. Each time PROCEDURE statement is encountered, it is entered in the IGBLK table. Each entry in IGBLK requres two words of IBM 7044 which consists the following:

Procedure name
SPTGO

where 'SPTGO' is the starting pointer in the GOTO table wherefrom any external label needed for this procedure block is put. The first entry in the GOTO table for any procedure block is the procedure name itself and the string of unfound labels, needed by that particular procedure, are put thereafter. If there is no global label needed by this procedure block, SPTGO is set to zero.

'NPB' is the pointer for handling IGBLK table. When an END statement is encountered the 'NPB' is moved up.

THE SYMBOL TABLE

During the Pass 1 of the compilation of a source programme, Symbol table entries are filled. A Symbol table entry consists of four consecutive words packed with information concerning one of the following entities which occur in the programme segment (External Procedure) being compiled:

(1) Variable name

    (i) Non subscripted

    (ii) Single subscripted

    (iii) Double subscripted

    (iv) Triple subscripted

(2) Procedure name

(3) Label name

Symbol Table Entry for

    (a) Variable name: Location

MODE	irpn	TYPE	nl		t

NAME		$t+1$

VARNO	INV	EVB		$t+2$

P	INC	Q		$t+3$

nl = pointer to next item with the same hash address
if there is any. Else nl = 0.

TYPE = 0   for non subscripted variables

= 1   for single subscripted variables.

= 2   for double subscripted variables.

= 3   for triple subscripted variables.

= 4   for Procedure name.

= 5   for label name.

EVB: This is the Symbol table address of the block under which this name (variable, procedure or label) has come. EVB=0 for the External Procedure name.

MODE = 1   for attribute FIXED.

= 2   for attribute BIT.

= 3   for attribute CHARACTER.

irpn = Internal relative position number of the variable in the block under which this variable name has been declared.

VARNO= Internal sequence number of the subscripted variable.

= 0   for non subscripted variable.

NW   = Number of words needed by the variable.

P   = Upper limit of the first subscript.

= 0   for non subscripted variable.

Q   = Upper limit of the second subscript.

= 0   for single and non subscripted variable.

NC   = Number of characters in the variable name itself, e.g. if the variable is 'ABD' then NC=3.

(b) Procedure name:

MODE	IIN	4	NL

Name

	NUM2		EVB

NUM1		NC	NP

MODE = 1   for simple procedure without any formal parameter.

  = 2   for parametric procedure.

  = 3   for functional procedure with attribute FIXED. ·

  = 4   for functional procedure with attribute BIT.

  = 5   for functional procedure with attribute CHARACTER.

NN = Number of bits needed for a BIT or CHARACTOR attribute
  of Built in function or Home made function.

  = 0   otherwise.

NUM2 = Level of the block.

NP = No. of formal parameters in the procedure if any.

  = 0   otherwise.

(c) Label name:

Label name includes DO name and BEGIN name also.

MODE	NN	5	NL

Name

		EVB

	NC	NUM3
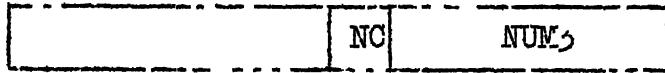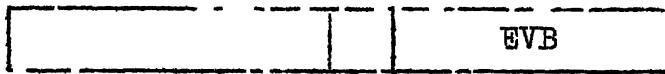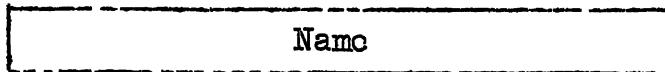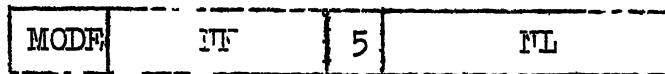
MODE = 0 if this label is under Procedure block.

      = 1 if it is under Begin block.

      = 2 if it is under DO block.

NN   = Level sequence number.

NUM3 = Internal DO number if this label is under DO block.

Here an example is taken to illustrate the coding of the source statement (IITPL).

```
 ~PL/1
 P1..PROCEDURE OPTIONS (MAIN) ,.
      DECLARE (A,B,C,D,E,I) FIXED ,.
      DECLARE F(5,6,7) FIXED ,.
   LI..A=A+B ,.
   L..IF(A.EQ.B) THEN IF(C.GT.2) THEN IF(A.NE.C)THEN A=0,.
      ELSE ,.
      ELSE IF(A.LT.8) THEN C=5 ,.
      ELSE A=C ,.
      CALL P2(A) ,.
      DO I=1, D WHILE A=3, 4 TO 7 BY B ,.
      A=A*(B+C/D**5)-8 ,.
      END/* THIS SHOWS THE END OF DO GROUP */ ,.
      GET EDIT (C,D,E) (F (6), X(3)) ,.
   B1..BEGIN ,.
      DECLARE (A,M,N,P) FIXED ,.
      A = A+B ,.
      END B1 ,.
   P2..PROCEDURE (G) /* SECOND BLOCK STARTS */ ,.
      DECLARE G FIXED ,.
      GOTO M,.
      RETURN ,.
      END P2 ,.
```

```
G..FORMAT(3 F(6),X(1)) ,.
    PUT EDIT (((F(A,B,C) DO A=D TO E
    BY I) DO B=3 TO E) DO C=I TO 6
    BY 3) (R(G)) ,.
    END P1 ,.
    *DATA
```

Coding (Final)

```
$IBMAP P1

P1      TRA     .I0001
.V0001  SXA     ..IDX2,2
        TSX     .SUBR3,4
        PZE     7
        PZE     5
        PZE     6
        PZE     7
.I0001  BSS
        CLA     =0̸000001000330
        TSL     .RTRTN
M       CLA     1,1
        ADD     2,1
        STO     1,1
```

L..IF(A.EQ.B) THEN IF(C.GT.2) THEN IF(A.NE.C) THEN A=0,.

```
           L      CLA      1,1
                  SUB      2,1
                  TSX      RO.EQ.,4
                  TZE      .E0001
                  CLA      3,1
                  SUB      =2
                  TSX      RO.GT.,4
                  TZE      .E0002
                  CLA      1,1
                  SUB      3,1
                  TSX      RO.NE.,4
                  TZE      .E0003
                  CLA      =0
                  STO      1,1
                  TRA      .F0001
ELSE,.
-
ELSE IF(A.LT.8) THEN C=5,.

ELSE A=C ,.

ELSE ,.

           .E0003 TRA      .F0001
           .E0002 BSS
                  CLA      1,1
                  SUB      =8
                  TSX      RO.LT.,4
```

```
                    TZE         .E0004
                    CLA         =5
                    STO         3,1
                    TRA         .F0001
          .E0004 BSS
                    CLA         3,1
                    STO         1,1
                    TRA         .F0001
          .E0001 TRA           .F0001
          .F0001 BSS


CALL P2(A),.         TSL         P2
                    TXI         *+3,,1
                    PZE         1,3,mode
                    TRA         M

DO I = 1,D WHILE A=3,4 TO 7 BY B ,.
A = A^(B+C/D**5)-8 ,.
END /* THIS SHOWS THE END OF DO GROUP */ ,.
                    CLA         =1
                    STO         6,1
                    TSL         .D0001
                    CLA         4,1
                    STO         6,1
                    CLA         1,1
                    SUB         =3
                    TSX         RO.EQ.,4
                    TZE         *+2
```

```
          TSL       .D0001
          CLA       =4
          STO       6,1
          CLA       =7
          STO       D.0001+3
          CLA       2,1
          STO       D.0001+4
.W0001    TSL       .D0001
D.0001    TSX       .DORTN,4
          CLA       6,1
          STO       6,1
          PZE       0
          PZE       0
          TZE       E.0001
          TRA       .W0001
D0001     TRA       -X
```

* Coding of next statement starts here i.e.
Assignment statement in this case.

```
          LDQ       4,1
          MPY       4,1
          STQ       .T.+0
          MPY       .T.+0
          MPY       4,1
          STQ       .T.+0
          LDQ       3,1
```

```
                    PXD     ,0

                    DVP     .T.+0

                    LLS     35

                    ADD     2.1

                    LRS     35

                    MPY     1,1

                    LLS     35

                    SUB     =8

                    STO     1,1

                    TRA     .D0001

          E.0001    BSS

Coding for all      TSL     STNDX.
the rest statements:
                    TSX     TSHIO.,4

                    TWO     FILO5.

                    PZE     nS

                    TSL     RESET.

                    TSL     HNLIO.

                    STO     3,1

                    TSL     HNLIO.

                    STO     4,1

                    TSL     HNLIO

                    STO     5,1

                    TSX     RTNIO.,4

                    TRA     mE

          nS        TSX     IOHIC.,4

                    PZE     6

                    TSX     IOHXC.,4

                    PZE     3

                    TRA     IOHEF.

          mE        BSS
```

```
B1       TRA      .I0002
.I0002  BSS
         TSL      B.0002
         CLA      1,1
         ADD*     5,1
         STO      1,1
         CLA      =2
         TSL      .RTRTN
         TRA      .B0002
B.0002  TRA      **
         CLA      =Ø000002000005
         TSL      .RTRTN
         CLA      =3
         LDQ      =1
         TSL      .RTRTN
         ADD      =2
         STO      5,1
         TRA      B.0002
.B0002  BSS
         TRA      .P0003
P2       TRA      **
         LAC      *-1,4
         TRA      .I0003
.I0003  BSS
         CLA      =Ø000003400001
         TSL      .RTRTN
```

```
         TXI     *+2,,1
         PZE     1
         CLA     1,4
         PLT     1,4
         ANA     =0000000077777
         ANA     =0000060277777
         ORA     =0050000000000
         SLW     *+1
         CLA     **
         STO     1,1
         TSL     B.0003
         TRA     .L0001
         TRA     A.0003
         TRA     .R0001
.L0001   CLA     =2
         ISL     .RTRTN
         LXA     P2,4
         SXA     *+1,4
         LXA     **,4
         TIX     *+1,4,1
         SXA     *+1,4
         TRA     **
.R0001   BSS
A.0003   LAC     P2,4
         CLA     1,4
         PLT     1,4
         ANA     =0000000077777
         ANA     =0000060277777
         ORA     =0050000000000
```

```
              SLW      *+2
              CLA      1,1
              STO      *˅
              CLA      =2
              TSL      .RTRTN
              TRA      P2
B.0003 TRA      **
              TRA      B.0003
.P0003 BSS
              TRA      nE
G       AXT      3,2
              TSX      IOHIC.,4
              PZE      6
              TSX      IOHXC.,4
              PZE      1
              TRA      IOHEF.
nE      BSS
              TSL      STNDX.
              TSX      STHIO.,4
              TWO      FIL06.
              PZE      2S
              TSL      RESET
              LDQ      6,1
              STQ      3,1
.S0002 BSS
              LDQ      .   =3
              STQ      2,1
```

```
        CLA     5,1
        PAX     0,4
        SXD     .S0003,4
.S0004  BSS
        LDQ     4,1
        STQ     1,1
        CLA     5,1
        PAX     0,4
        SXD     .S0005,4
        CLA     6,1
        PAX     ,4
        SXD     .T0003+2,4
        LDQ     3,1
        VIM     =983040,,15
        ALS     15
        LDQ     2,1
        VMA     =165840,,15
        ADD     1,1
        SUB     ..IDX1
        PAC     ,2
        CLA     6,1
        PAX     ,4
        SXD     .N0001,4
.S0006  BSS
        TSX     .V0001,4
        ONE     1,1
        ONE     2,1
        ONE     3,1
        CLA     -29,2
```

```
              TSL      HNLIO.
     .N0001 TXI       *+1,2,**
     .T0003 CLA       1,1
              PAX      0,4
              TXI      *+1,4,**
              PXA      0,4
              STO      1,1
     .S0005 TXL       .S0006,4,**
     .T0002 CLA       2,1
              PAX      0,4
              TXI      *+1,4,1
              PXA      0,4
              STO      2,1
     .S0003 TXL       .S0004,4,**
     .T0001 CLA       3,1
              PAX      0,4
              TXI      *+1,4,3
              PXA      0,4
              TXL      .S0002,4,6
              TSX      RTNIO.,4
              LXA      ..IDX2,2
2S       EQU      G
              TRA      S.JXIT
              ENTRY    ..VAR
              EXTERN   ..IDX2
```

```
          EXTERN   .RORTN

          EXTERN   .RTRTN

          EXTERN   .SUBR3

          EXTERN   .DORTN
..VAR     BSS      223
.T.       BSS      1
          END      P1
$ENTRY
```

1971-M-SIN-PAS